CS 310 | Spring 2022 | Unit 4 | Survey of Functional Languages

Topics: *Functional programming, language translation, referential transparency, recursion*

Follow the instructions using source code. Submit the indicated files (📎) via the assignments in the eCampus lecture section by the due dates on the course calendar. For each submitted file, include your full name as author and a statement acknowledging that your work complies with the academic integrity policy.

If you are pair programming, you and your peer each submit the indicated files independently and are graded independently. For each submitted file, also include your peer's full name as co-author and a description of your and your peer's contributions.

INSTRUCTIONS

FUNCTIONAL LANGUAGE TRANSLATION

A Choose **one** of the five **Scheme** functions below.

1 If you are pair programming, you and your peer can choose the same or different functions.

```
(define (append lizt1 lizt2)
  (cond
    ((null? lizt1) lizt2)
    (else (cons (car lizt1)
      (append (cdr lizt1) lizt2)))
  )
)
```

```
(define (map fun lizt)
  (cond
    ((null? lizt) '())
    (else (cons (fun (car lizt))
      (map fun (cdr lizt))))
  )
)
```

```
(define (member atm lizt)
  (cond
    ((null? lizt) #f)
    ((eq? atm (car lizt)) #t)
    (else (member atm (cdr lizt)))
  )
)
```

```
(define (same lizt1 lizt2)
  (cond
    ((null? lizt1) (null? lizt2))
    ((null? lizt2) #f)
    ((eq? (car lizt1) (car lizt2))
      (same (cdr lizt1) (cdr lizt2)))
    (else #f)
  )
)
```

```
(define (intersect lizt1 lizt2)
  (cond
    ((null? lizt1) '())
    ((memq (car lizt1) lizt2)
      (cons (car lizt1) (intersect (cdr lizt1) lizt2)))
    (else (intersect (cdr lizt1) lizt2))
  )
)
```

B Reimplement the function in 📎 **Clojure**, a modern dialect of Lisp (**20%**).

1 You must complete this reimplementation individually, even if you are pair programming.

C Reimplement the function in 📎 **Haskell,** a purely functional language with conventional syntax (**40%**).

1 If you are pair programming, you can complete this reimplementation with a peer or individually.

D Reimplement the function in 📎 **Scala**, a multi-paradigm functional language with Java-like syntax (**40%**).

1 If you are pair programming, you can complete this reimplementation with a peer or individually.

E For **15% bonus** on the unit, reimplement the function in 📎 **APL**, a functional language with terse and unconventional syntax.

1 If you are pair programming, you can complete this reimplementation with a peer or individually.

F Each reimplementation must define the same protocols as the original Scheme implementation.

1 Names may be altered when they conflict with reserved words or library functions (ex. `lizt` instead of `list`).

2 Generic types should be used when possible, otherwise consistent non-generic types should be used instead.

G Each reimplementation must produce the same semantics as the original Scheme implementation when possible.

1 No library function may be called which is equivalent to the function being reimplemented.

2 No iteration may be used, only recursion.

3 There must be referential transparency, thus no side effects.

4 When two values are compared for equality, any suitable equality semantics may be used.